



eduresourceza.com

Section 1 - Information Representation and Multimedia
1.1 - Data Representation

The Key Terms:

- Binary: It is a base two number system based on the values 1 and 0 only
 - Bit: Bit stands for binary digit
 - One's Complement: each binary digit in a number is reversed to allow both negative and positive numbers to be represented
 - Two's Complement: each binary digit is reversed and 1 is added in the right-most position to produce another method of representing positive and negative numbers
 - Sign and Magnitude: binary number system where the left-most bit is used to represent the sign (0 = + and 1 = -). The remaining bits represent the binary value
 - Hexadecimal: a number system based on the value 16
 - Memory Dump: content of a computer memory output to screen or printer
 - Binary Coded Decimal (BCD): number system that uses 4 bits to represent each binary digit
 - ASCII Code: Coding system for all the characters on a keyboard and control codes
 - Character Set: A list of characters that have been defined by computer hardware and software. A method of coding is required for computers to understand human characters
 - Unicode: Coding system which represents all the languages of the world
-

Denary Number System: This system uses the digits 0 to 9 which are placed in weighted columns. (eg 10000, 1000, 100, 10)

Binary Number System: This system allows only two values, which are 0 and 1. It is adopted by designers of computer systems and is the building block in all computers.

- A switch in the ON position is represented by 1
 - A switch in the OFF position is represented by 0
 - Each binary digit is known as a bit
-

Converting from Binary to Denary:

128	64	32	16	8	4	2	1
1	1	1	0	1	1	1	0

$$128 + 64 + 32 + 8 + 4 + 2 = 238 \text{ (denary)}$$

Converting from Denary to Binary:

- Consider converting the denary number 107 to binary
- This involves placing 1's in the correct positions so that the total is 107

128	64	32	16	8	4	2	1
0	1	1	0	1	0	1	1

Binary Addition and Subtraction:

The methods used to represent positive and negative numbers:

- One's complement
- Two's complement

What happens in one's complement?

In one's complement, each digit in the binary number is inverted (0 becomes 1, 1 becomes 0)

- 0101 1010 becomes 1010 0101 (90 → -90)

What happens in two's complement?

In two's complement, each digit in the binary number is inverted and a '1' is added to the right-most bit.

- $1010\ 0101 + 1 \rightarrow 1010\ 0110$
- Note that $1 + 1 = 0$, a carry of 1

The two's complement uses these weightings for an 8-bit number representation:

-128	64	32	16	8	4	2	1
1	1	0	1	1	0	1	0
0	0	1	0	0	1	1	0

- The first example is $-128 + 64 + 16 + 8 + 2 = -38$
- The second example is $32 + 4 + 2 = 38$

Note: For using two's complement, remember to invert the digits and then add 1

Binary Addition:

Example 1.1: Add 0010 0101 (37 in denary) and 0011 1010 (58 in denary)

Solution:

-128	64	32	16		8	4	2	1
0	0	1	0		0	1	0	1
				+				
0	0	1	1		1	0	1	0
				=				
0	1	0	1		1	1	1	1

This gives us 0101 1111 which is 95 in denary (correct answer)

Example 1.2: Add 0101 0010 (82 in denary) and 0100 0101 (69 in denary)

Solution:

-128	64	32	16		8	4	2	1
0	1	0	1		0	0	1	0
				+				
0	1	0	0		0	1	0	1
				=				
1	0	0	1		0	1	1	1

This gives us 1001 0111 which is -105 in denary. The addition of positive numbers resulted in a negative answer.

Why is the answer negative?

This is because the result of the addition has produced a number which is outside the range of values which can be represented by the 8-bits being used. (the addition gives 151, which is more than 127). As a result, this causes overflow.

Note: The largest value represented by a binary number is +127

Binary Subtraction:

Example 1.3: Carry out the subtraction 95 - 68 in binary

Solution:

1. Convert the two numbers into binary

a. 95 = 0101 1111

b. 68 = 0100 0100

2. Find the two's complement of 68

a. Invert the digits: 1011 1011

b. Add 1: 1011 1011 + 1 →
1011 1100

3. Add 95 and -68:

	-128	64	32	16		8	4	2	1
	0	1	0	1		1	1	1	1
					+				
	1	0	1	1		1	1	0	0
					=				
1	0	0	0	1		1	0	1	1

The additional 9th bit is simply ignored, giving the binary number 0001 1011 (denary number 27)

Example 1.4: Carry out the subtraction 49 - 80 in binary

Solution:

1. Convert the two numbers into binary

a. 49 = 0011 0001

b. 80 = 0101 0000

2. Find the two's complement of 80:

a. Invert the digits: 1010 1111

b. Add 1: 1010 1111 + 1 →
1011 0000

3. Add 49 and -80:

This gives us 1110 0001, which is -31 in denary (correct answer)

-128	64	32	16		8	4	2	1
0	0	1	1		0	0	0	1
				+				
1	0	1	1		0	0	0	0
				=				
1	1	1	0		0	0	0	1

Measurement of the size of computer memories:

What is a byte?

A byte is the smallest unit of memory in a computer. Computers use largery bytes, such as 16-bit and 32-bit systems, but they are multiples of 8.

Name of memory size	Equivalent Denary Value (bytes)
1 kilobyte (1 KB)	1,000
1 megabyte (1 MB)	1,000,000
1 gigabyte (1 GB)	1,000,000,000
1 terabyte (1 TB)	1,000,000,000,000
1 petabyte (1 PB)	1,000,000,000,000,000

Note: This system of numbering is technically inaccurate, because it is based on the SI system of units where 1 kilo is equal to 1000.

How is memory size measured?

Memory size is measured in powers of 2

IEC - International Electrotechnical Commission

Name of memory size	Number of bytes	Equivalent denary value (bytes)
1 kibibyte (1 KiB)	2^{10}	1,024
1 mebibyte (1 MiB)	2^{20}	1,048,576
1 gibibyte (1 GiB)	2^{30}	1,073,741,824
1 tebibyte (1 TiB)	2^{40}	1,099,511,627,776
1 pebibyte (1 PiB)	2^{50}	1,125,899,906,842,624

Note: This system is more accurate. Internal memories such as RAM should be measured using the IEC system

Hexadecimal Number System:

Hexadecimal Number System: It is closely related to the binary system. Hexadecimal is a base 16 system.

- The numbers 0 to 9 and A to F are used to represent hexadecimal digits
- A = 10, B = 11, C = 12, D = 13, E = 14 and F = 15

Weightings of Hexadecimal System:

1,048,576	65,536	4,096	256	16	1
16^5	16^4	16^3	16^2	16^1	16^0

Note: Since $16 = 2^4$, four binary digits are equivalent to one hexadecimal digit

Converting from binary to hexadecimal and hexadecimal to binary:

Example 1.5: Convert 1011 1110 0001 from binary to hexadecimal

Solution:

1. First split it into groups of 4-bits:
 - a. 1011 1110 0001
2. Then find the equivalent hexadecimal digits:
 - a. B E 1

Example 1.6: Convert 10 001 1111 1101 from binary to hexadecimal

Solution:

1. First split it into groups of 4-bits:
 - a. 10 0001 1111 1101
2. The left group only contains 2-bits, so add in two 0s to the left:
 - a. 0010 0001 1111 1101
3. Now find the equivalent hexadecimal digits:
 - a. 2 1 F D

Example 1.7: Convert this hexadecimal number to its binary equivalent 45A

Solution:

8	4	2	1		8	4	2	1		8	4	2	1
0	1	0	0		0	1	0	1		1	0	1	0

Example 1.8: Convert this hexadecimal number to its binary equivalent BF08

Solution:

8	4	2	1		8	4	2	1		8	4	2	1		8	4	2	1
1	0	1	1		1	1	1	1		0	0	0	0		1	0	0	0

Uses of the Hexadecimal System:

Memory Dumps:

- Hexadecimal is often used when developing new software or trying to trace errors in programs
- When the memory contents are output to a printer or monitor, it is known as memory dump

Advantages of Hexadecimal System:

1. Allows program developers to find errors in code
2. Hexadecimal is more manageable than binary
3. It is a power fault-tracing tool

Note: Requires knowledge of computer architecture to be able to interpret results

Binary Coded Decimal (BCD) System:

BCD System: The BCD system uses a 4-bit code to represent each denary digit

Binary	Denary	Binary	Denary
0000	0	0101	5
0001	1	0110	6
0010	2	0111	7
0011	3	1000	8
0100	4	1001	9

The BCD format of denary number 3165:

3165 → 0011 0001 0110 0101

The different ways to store 4-bit codes:

1. It can be stored as a half-byte
2. It can be stored as two 4-bit codes together to form one byte

Method 1:

0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	1
0	0	0	0	0	1	1	0
0	0	0	0	0	1	0	1

Method 2:

0	0	1	1	0	0	0	1
0	1	1	0	0	1	0	1

Uses of BCD:

The uses of BCD:

1. Representation of digits on a calculator
2. Representation of digits on a clock display

Why do exact values need to be stored for accounting/monetary values?

Exact values need to be stored to prevent significant errors from accumulating. Monetary values used a fixed-point notation.

Consider adding \$0.37 and \$0.94 together using fixed-point decimals:

0	0	0	0	0	0	0	0	.	0	0	1	1	0	1	1	1
								+								
0	0	0	0	0	0	0	0	.	1	0	0	1	0	1	0	0

Using binary addition, this sum will produce 0000 0000.1100 1011:

- This gives 1100 (denary 12) and 1011 (denary 11), which are incorrect, because 12 and 11 are not denary digits

Note: The solution to this problem is to add 0110 (denary 6) whenever a problem comes. The computer can be programmed to add 0110.

Adding 0.07 to 0.04 using the solution:

	0	1	1	1
		+		
	0	1	0	0
		=		
	1	0	1	1
		+		
	0	1	1	0
		=		
1	0	0	0	1

Note: The fifth bit is carried to the next decimal position

Adding 0.3 to 0.9 using the solution:

0	0	1	1
	+		
1	0	0	1
	+		
			1
	=		
1	1	0	1

Note: This produces 1101, which isn't a denary digit, therefore it is flagged as an error and 0110 is added

	1	1	0	1
		+		
	0	1	1	0
		=		
1	0	0	1	1

Adding 1 to 0000 0000 produces:

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

Final Answer:

0	0	0	0	0	0	0	1	.	0	0	1	1	0	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Note: This is 1.31 in denary, which makes it the correct answer

ASCII Codes and Unicodes:

ASCII → American Standard Code for Information Interchange

ASCII Code: Used in communication systems and computer systems

- The standard ASCII code character set consists of 7-bit codes (0 to 127 denary codes or 0 to 7F in hexadecimal)

a	1	1	0	0	0	0	1
A	1	0	0	0	0	0	1
y	1	1	1	1	0	0	1
Y	1	0	1	1	0	0	1

- Notice the sixth bit changes from 1 to 0 when comparing uppercase and lowercase characters

Note: Extended ASCII uses 8-bit codes to allow non-English characters and for drawing characters to be included.

Why has Unicode been developed?

Because ASCII code has a number of disadvantages and unsuitable for different purposes

The similarities and differences between ASCII Code and Unicode:

- Unicode has overlap with ASCII code because the first 128 characters are the same
- Unicode can support several thousand different characters
- ASCII uses 1 byte to represent a character, however Unicode can support 4 bytes per character

What were the goals of Unicode?

- Create a universal standard that covered all languages
- Produce a more efficient coding system than ASCII